AmbientDB: P2P Data Management Middleware for Ambient Intelligence

Willem Fontijn
Philips Research
willem.fontijn@philips.com

Peter Boncz
CWI
boncz@cwi.nl

Abstract

The future generation of consumer electronics devices is envisioned to provide automatic cooperation between devices and run applications that are sensitive to people's likings, personalized to their requirements, anticipatory of their behavior and responsive to their presence. We see this 'Ambient Intelligence' as a key feature of future pervasive computing. We focus here on one of the challenges in realizing this vision: information management. This entails integrating, querying, synchronizing and evolving structured data, on a heterogeneous and ad-hoc collection of (mobile) devices. Rather than hard-coding data management functionality in each individual application, we argue for adding high-level data management functionalities to the distributed middleware layer. Our AmbientDB P2P database management system addresses this by providing a global database abstraction over an ad-hoc network of heterogeneous peers.

1. Introduction

Future generations of consumer electronic devices will make computing power and connectivity omnipresent, yet hidden from the users view. This pervasive computing infrastructure will be used to create an environment that anticipates users wishes instead of just responding to direct commands. The aim is to improve quality of life by creating the desired atmosphere and functionality via personalized, interconnected systems and services. This vision, called *Ambient Intelligence* (AmI), is the focal point of much ongoing research [1]. The 'Ambient' part of AmI refers to the unobtrusiveness of the technology, both physically, by embedding it in the environment, and functionally, by making user interaction mostly implicit. The latter will entail, for instance, habit watching [3]. The 'Intelligence' part of AmI refers to the way the system integrates and relates the data from a wide variety of sources to create the perception of intelligence. The sources range from simple sensor nodes to Personal Video Recorders (PVRs) with sophisticated preference-based meta data. Perceived intelligence implies that the AmI system has to present to the user a unified and consistent view irrespective of the context and location of the user or the type of interaction. In an ad-hoc mobile environment, this requires novel synchronization procedures, transparent to the user [4].

The *Connected Home* [5] may be seen as a first step towards AmI. Devices contain their own embedded DBMS, and operate in isolation below the network layer (see Fig.1a). If one device requires information located on another device it will first have to find the latter device and query it for the information. When the complexity of the network increases, i.e. more and more diverse devices are introduced, the increasing number of possible combinations will make it hard to create robust applications. Also, the performance of such a system will degrade rapidly. This could be countered by pre-emptive data aggregation on resource-rich central servers, but this has multiple drawbacks. It creates mounting overhead as the complexity increases, implies that certain functionality is available only when in range of a server and makes the system vulnerable to point failures. Finally, it is questionable from a marketing viewpoint whether customers will buy an expensive home-server that just improves the performance of other devices.

An alternative to the central server approach is to keep the participating devices fully autonomous but cooperative. A collection of such devices we call a *Device Society*. Each device has its own responsibility to deliver certain functionality and the collection as a whole delivers the environment to run high-level applications. In such a system, most data is stored distributed and only integrated at query time. Such distributed storage is robust and

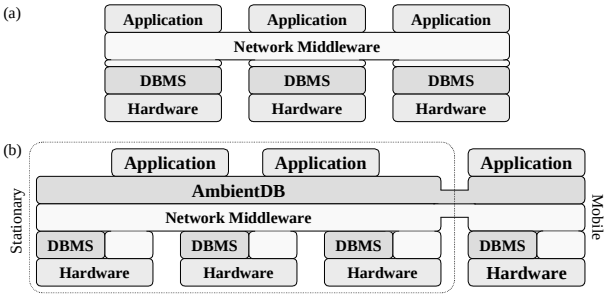

Figure 1: Concept of (a) Connected Home, (b) Ambient Intelligence environment

scalable but data management is not easy anymore.
In this paper we discuss a data management approach for the latter strategy and present a prototype system. We first illustrate its requirements using a scenario. In Section 2.2 a technical realization is presented.

1.1. Scenario: Music Playlist Generation

Consider the problem of playing music for a user that is appropriate to his context and state irrespective of his location. All music owned by this particular user, as well as meta data describing this music is stored in a music database distributed over one or more devices, such as portable music (mp3) players, Smart Phone, PVR and/or PC. To the user, this music collection should appear conceptually as a single collection.
Three autonomous processes are active on this music database. The first aims to improve the *profile* of the user. The user can actively rate pieces of music, indicating which songs he likes or dislikes. This *explicit rating* is useful to establish a first draft profile but requires direct user interaction. A more convenient way to improve the profile is *implicit rating*. The system logs the reaction of the user to different pieces of music in different contexts and the logs are later processed to derive preferences of the user in various contexts.
The second process aims to extend the music collection. The profile of the user may be compared to profiles of others, using collaborative filtering [3]. If one user likes many songs another user likes, the first may also like songs unknown to him that the second user likes.
The third process recommends music. Based on the context, the user is offered a selection of his total music collection. The context of the user may be derived from many clues, e.g. ambient lighting, time of day and facial expression. If the user leaves the house and takes a portable music store with him, the system assists him in selecting the music subset that will probably meet his musical needs for the particular trip. While away, the portable music device records implicit rating cues and may pick up some additional pieces of music.
This system should not be static but able to evolve. If the user buys a watch that senses his skin temperature, the music recommender should discover that by combining this new context information with other data, it could gauge the mood of the user. From that moment on, the recommender takes mood into account.

1.2. The idea: database middleware

Our challenge of managing a sea of evolving data sources among possibly large and dynamic Device Societies should be addressed in the middleware layer (see Fig. 1b). Middleware services already provide the basic infrastructure for application integration [9], integrating not only data but also a wide variety of context sources [6], providing applications with information about network and device resources, and allowing applications to reconfigure when conditions change [5]. While there have been middleware systems with some data management support [8][9] we aim to raise this to the level of full DBMS functionality.
By putting such DBMS functionality inside or on top of the middleware layer, all data sources are virtually merged, shielding applications from the underlying complexities. Applications release their queries and get the results as if accessing a single database system. Events such as the integration of new devices or data sources, or failure of devices, and the creation of new data types, can in great part be handled as schema evolution in this database. Accepting queries in a high-level language that describes *what* data an application needs instead of *how* to exactly obtain its results, allows a *query optimizer* rather than the application programmer to automatically find an efficient solution for executing a query, taking into account indexing structures, system and network load conditions and concurrent requests. Also, this provides *data independence*, meaning that the data representation can change over time without this breaking the applications using the data. These are the classic database advantages that we are now able to bring into the pervasive computing domain. Finally, this approach enables evolutionary introduction of new functionality. By supplementing middleware for the Connected Home with DBMS functionality, we create a breeding ground for more and more sophisticated and AmI applications.

2. AmbientDB: a P2P DBMS

The goal of our AmbientDB system [10] is to provide full relational database functionality for standalone operation in autonomous devices that may be mobile and disconnected for long periods of time, while enabling them to cooperate in an ad-hoc way with (many) other AmbientDB devices. Hence, our choice for P2P, as opposed to designs that use a central server. AmbientDB uses ‘abstract’ tables, i.e. applications are ignorant of where data resides. Internally, a table may be private to the node, or distributed over many nodes in the network. The actual content of a distributed table is formed by the union of table partitions in all nodes that are connected at that time.

2.1. Key Functionalities

Since our work touches upon many sub-fields of database research [7], we highlight the main differences.

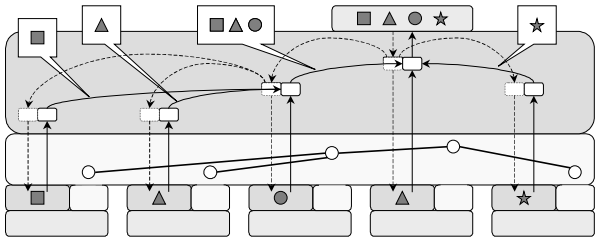

Figure 2: Concept of AmbientDB. The application on top issues a query to the AmbientDB layer that is propagated (dashed line) to all connected peers. The query result (solid line) is aggregated along the query path and presented to the application. The binary tree in the network layer represents the network topology.

Distributed database technology presupposes that the collection of participating sites and communication topology is known a priori. AmbientDB does not. *Federated database technology*, the current approach to heterogeneous schema integration, focuses on statically configured combinations of databases instead of ad-hoc Device Societies. *Mobile database technology* generally assumes that mobile nodes are (weaker) clients that synchronize with a centralized database server over a narrow channel. Again, AmbientDB does not. Finally, *P2P file sharing* systems do support decentralized, ad-hoc Device Societies, but allow only simple keyword text search (as opposed to structured database queries).

2.1.1. Self Organization

P2P technologies are able to adapt to changes in the environment and work without central planning. In order to provide efficient indexed lookup into its distributed database tables, AmbientDB makes use of Chord [11]. Chord is a Distributed Hash Table (DHT), a scalable P2P data structure for sharing data among a potentially large collection of nodes, allowing nodes to join and leave without making the network unstable. It uniformly distributes data over all nodes using a hash-function, enabling efficient *O(log(N))* data lookup. To improve scalability in situations where some devices are resource-poor, AmbientDB keeps devices out of Chord to prevent overloading them with data they cannot store or with queries they cannot handle. Upon connection, low-resource nodes transfer their data to a resource-rich neighbor that handles queries on behalf of them.

2.1.2. Query Processing

AmbientDB performs a three level query translation:
(1) abstract algebra: A user query is posed in the "abstract global algebra". This is a standard relational query language, providing the basic operators for selection, join, aggregation and sorting.
(2) concrete algebra: These are concrete strategies for resolving the basic relational operators. Typically, each abstract operator has multiple concrete variants. E.g., there is a broadcast-select, that executes a selection operator on a distributed table by flooding the network (broadcast) and collecting all matches. There is also a variant that exploits a Chord DHT index, which may be used if a global index on a table column was defined in the schema. Thus, many different concrete plans may exist for an 'abstract' query, and the query optimizer in AmbientDB is used to select a good plan.
(3) dataflow algebra: A very small kernel of basic operators is sufficient to implement the concrete algebra. Each concrete operator is mapped onto a *wave-plan* that consists of a graph of dataflow operators. Next to query processing the dataflow operators provide functionality for splitting and merging data streams.
We plan to augment AmbientDB with support for triggers, such that applications can be alerted to interesting events rather than poll the global database with queries [12].

2.1.3. Synchronization

The aim of traditional (distributed) database technology, to provide strict consistency, is not appropriate for P2P database systems. Algorithms, such as two-phase locking, are too expensive for a large and sparsely connected collection of nodes. Many applications do not need full transactional consistency, but just a notion of final convergence of updates. Also, applications often have effective conflict resolution strategies that exploit application-level semantics. Thus, the challenge for a P2P DBMS is to provide a powerful formalism in which applications can formulate synchronization and conflict resolution strategies. Our first target is to support applications that use rule-based synchronization expressed in a prioritized set of database update queries.

2.1.4. Schema Integration & Evolution

As devices differ in functionality and make, their data differs in semantics and form. We use table-view based schema integration techniques [13] to map local schemata to a global schema. AmbientDB itself does not address the automatic construction of such mappings, but aims at providing the basic functionality for applying, stacking, sharing, evolving and propagating such mappings. Providing support for schema evolution within one schema, e.g. such that old devices can cooperate with newer ones, is often forgotten. We foresee that a global certifying entity keeps track of changes in the various sub-schemas, maintaining bi-directional mappings between versions. Schema deltas are certified such that one peer

may carry it to the next, without need for direct communication with a centralized entity.

2.2. Scenario using P2P data management

The gist of our approach is that we believe that P2P data management functionality will make it easier to construct Ambient Intelligent applications. To illustrate how we see that happen, let us go back to the problem of managing and navigating music intelligently.

The schema created by the music player contains a LOG table where per-user song play counts are kept (see Fig. 3). This is a *distributed table*, which means that the music application sees the union of all (overlapping) horizontal fragments at all participating devices of that moment as one big table. All devices maintain local play-counts for each (artist,user) combination in this LOG table. The schema specifies an *index* on LOG.artist, so each LOG entry is replicated in a Chord DHT and distributed over all nodes of the Internet domain, using the Chord hashing scheme (see Fig. 3). This allows to quickly locate users that played a particular artist.

2.2.1. Self Organization Example

The family music collection -- typically in the order of a few thousands of songs -- is distributed among the Device Society owned by family members. Some of these devices may have access to the Internet. The music players with embedded AmbientDBs form a self-organizing P2P network, connecting the nodes in order to share all music content in the "home domain", and a second -possibly huge- P2P network consisting of all music players reachable via the Internet, among which only the meta-information is shared. The home domain may contain some very low-resource devices in terms of CPU and storage (e.g. phone) that are kept out of the Chord DHT. In the Internet domain, the number of on-line nodes maybe large and the number of songs huge.

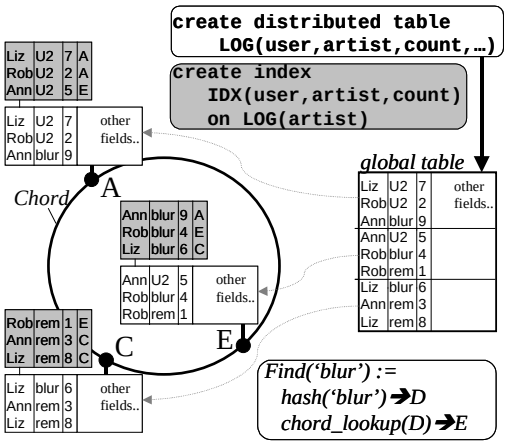

Fig.3: Scenario for sharing music metadata between many music players in the Internet domain. The distributed table LOG holds artist play-counts for each user. One can quickly find users that play a certain artist using a Chord index.

2.2.2. Schema Evolution Example

In our scenario, the user buys a watch with integrated body thermometer. This watch has Body Area Network (BAN) functionality (e.g. Bluetooth) such that it can communicate with the owner's phone or mp3 player when these are carried in his pocket. With the temperature meter watch comes an AmbientDB *schema update* that e.g. introduces a new TEMPERATURE table that stores (timestamp, temperature) records, and a *data propagation profile* with rules that specify the longevity of its records and a propagation strategy. Additionally, on a certified (vendor) site, the user community of the music player may store a *trigger update* that specifies a (complex) rule that derives a mood from the body temperature curve in conjunction with other personal characteristics stemming from other sources. When this mood indicates appreciation for the current song, an automatic playlist creation process is scheduled, aggregating songs similar to the one currently being played (this query is described in Section 2.2.4).

Note that schema updates may propagate in a P2P fashion (from watch to phone, from phone to home PC) or from a central Internet site, in any case though with a certifying mechanism. Also, schema updates may depend on a collection of sub-schema versions being present, such that during the next visit to the central vendor site, when the combination of music and temperature sub-schemas is detected, the user is alerted to the possibility of installing the "music-appreciation-trigger".

2.2.3. Update Propagation Example

The watch has a limited storage capacity, it can hold only a few records. Its synchronization rules, however, make it replicate temperature records to devices in the neighborhood (e.g. your Smart Phone). When it arrives at home, the Smart Phone then propagates these records to the home PC, where a health-monitoring agent might be running that periodically analyzes this log data using data mining techniques. The propagation rules may include a maximum lifetime that causes old records to be automatically deleted after e.g. a number of weeks.

2.2.4. Query Processing Example

The music player generates intelligent playlists, either because the user explicitly chooses an sample artist to generate a similar playlist from, or implicitly when the "music-appreciation-trigger" notices that you like an artist

being played. The two database queries below express a simple collaborative filtering method. The first query computes a relevance of other users' music taste from their play-count of an sample artist. The second query then computes a ranking by multiplying all artist play-counts of all users by the user relevance, and summing this per artist, returning a top-N.
We kept this example very simple for presentation purposes, but one can easily refine it, e.g. by increasing the granularity to songs (instead of artists) or making it work with a weighted collection of samples instead of one. The benefit is that for the application programmer, writing this kind of data-intensive applications on large ad-hoc networks is reduced to writing some relatively simple database queries. Database indices and query optimization then make sure that it runs efficiently without the application programmer having to worry about it [10].

```
% each record has the special field #nodeid
% that holds the device ID where it is stored

RELEVANT :=      % query will use Chord index
 SELECT user, SUM(playcount) AS relevance,
        #nodeid AS site
 FROM LOG
 WHERE artist = 'normalized artist name'
 GROUP BY user
 ORDER BY relevance DESCENDING LIMIT 5

SELECT L.artist AS artist,
       sum(L.playcount*R.relevance) AS score
FROM LOG L, RELEVANT R
WHERE L.user = R.user AND L.#nodeid = R.site
GROUP BY artist
ORDER BY score DESCENDING LIMIT 25
```

3. Current Status & Research Challenges

We hope to release a first version of AmbientDB early next year. We have focused so far on distributed query processing, and identified three functional levels that all require further research. On the top level, we need more experience with a wider variety of Ambient Intelligent applications to see what exact requirements they impose on a P2P DBMS. Also, if applications are to cooperate seamlessly, they need to operate in a compatible semantic framework. This is an "AI-hard" problem for the general case. We do see possibilities when trusted and standardized mappings are available. The second functional level is P2P data management. While we have a working query processor, it is likely that there are query execution algorithms that exploit the P2P architecture better. Also, loosely consistent or converging transactions as well as a schema mapping infrastructure remain open areas of research. The third functional level is P2P networking. P2P overlay technology often exhibits inefficient usage of physical resources, as these are opaque on the TCP overlay level. This could be improved by dynamic re-configuration of P2P networks, an important middleware research issue [5]. Better adaptation to device and network resources using e.g. slave- and super-nodes could be ways forward here.

4. Conclusions

Transparent distributed data management is crucial to Ambient Intelligent applications in Device Societies, and the P2P approach with AmbientDB as middleware offers a possible solution. It enables the creation of a high-level application development interface that is flexible and provides data independence, while taking the burden of data management optimization in a dynamic and ad-hoc distributed environment out of the hands of application programmers. The ability of AmbientDB to cope with adding devices and functionality dynamically provides for an evolutionary path for the introduction of Ambient Intelligence, thus alleviating one of the most prominent problems from a systems and marketing point of view.

5. References

[1] www.philips.com/research/ami
[2] www.semiconductors.philips.com/connected_home
[3] D. Nichols. Implicit Rating and Filtering, Proc. DELOS Workshop on Filtering and Collaborative Filtering, 1998.
[4] G. Montenegro. MNCRS: Industry Specifications for the Mobile NC, IEEE Internet Computing, 1998.
[5] M. Roman, F. Kon, R. Campbell. Design and Implementation of Runtime Reflection in Communication Middleware: the dynamicTAO Case. ICDCS Workshop on Middleware, 1999.
[6] A. Schmidt, M. Beigl, H.-W. Gellersen. There is more to context than location. Proc. of the Intl. Workshop on Interactive Applications of Mobile Computing, 1998.
[7] D. Kossmann. The state of the art in distributed query processing. ACM Computing Surveys, 32(4), 2000.
[8] G. Picco, A. Murphy, G.-C. Roman. On Global Virtual Data Structures. In: *Process Coordination and Ubiquitous Computing*, D. Marinescu, C. Lee, CRC Press.
[9] J. Carter, A. Ranganathan, S. Susarla. Khazana: An infrastructure for building distributed services. In Proc. Int. Conf. on Distributed Computing Systems (ICDCS'98), 1998.
[10] P. Boncz, C. Treijtel. AmbientDB: relational query processing in a P2P network. Proc. Workshop On Databases, Information Systems and P2P Computing (at VLDB'03), 2003.
[11] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. Proc. SIGCOMM Conf., 2001.
[12] J. Widom, S. Ceri: Active Database Systems: Triggers and Rules For Advanced Database Processing. Morgan Kaufmann.
[13] A. Halevy: Answering queries using views: A survey. VLDB Journal 10(4): 270-294, 2001.